Equivalences on program schemes

JOURNAL OF COMPUTER AND SYSTEM SCIENCES: 4, 205–219 (1970)

Equivalences on Program Schemes*

R. MILNER

Department of Computer Science, University College of Swansea, Swansea, GLAM, Wales

Received August 29, 1969

A program scheme may be informally described as a program with the interpretation (i.e., the meaning of the basic instructions or standard functions) left unspecified. This paper studies the equivalence of program schemes under different classes of interpretations, with emphasis on those in which the functions are permitted to be partial. Several different equivalence relations are defined, and their interrelationship and solvability examined both for the class of all program schemes, and for each subclass ($n \geqslant 1$) in which the number of registers is at most n.

1. INTRODUCTION

There is a growing interest in the rigorous proof of properties of computer programs. These properties depend on two distinct factors: (a) the primitives of the program (in machine code, the meanings of the basic instructions; in a high level language, the standard functions), and, (b) the *shape* (of the flow chart) of the program. If we define, informally for the moment, a *semiinterpretation* to consist of a universe of objects together with a certain definite meaning for each program primitive as a function on this universe, and an *interpretation* to consist of a semiinterpretation together with a definite set of input values (from the universe), then we can isolate those properties of a program which are dependent on (b) and not on (a) by demanding that they be invariant under change of interpretation. It turns out that such properties are rich enough to deserve separate study.

Another way to state this is to say that we are interested in the properties of *program schemes*, defined informally as "programs without interpretations." One such property is "equivalence under all interpretations." More precisely, Luckham, Park, and Paterson [3, 5] and Kaplan [2] (who uses the term *elemental program* for *program scheme*) say that two program schemes P, Q are *strongly equivalent* if under any interpretation (which of course includes a set of input values) *either* P and Q compute the same result *or* both fail to terminate. The above authors also show that, as so defined, *both* strong equivalence *and* its negation are not partially decidable binary

* This work was carried out under a grant from the Science Research Council.

predicates, where, as throughout this paper, "the n-ary predicate φ is *partially decidable*" means "$\{\langle x_1, x_2, \ldots, x_n\rangle \mid \varphi(x_1, x_2, \ldots, x_n)\}$ is recursively enumerable."

By contrast, Manna [4] (who uses *abstract program* for program scheme) defines P, Q *equivalent* to mean that under all interpretations they *both* terminate *and* compute the same results, and by showing how to effectively construct from P and Q a formula of first-order logic that is valid iff P and Q are equivalent (with this definition), he proves that this type of equivalence *is* a partially decidable predicate. We can note in passing that Manna's equivalence is properly stronger than the previously defined strong equivalence. Also the nonpartialdecidability of strong equivalence ensures that, for it, there can be no corresponding formula of first order logic. However, Cooper [1] constructs from any P, Q a formula of second-order logic which he shows is valid iff P, Q are strongly equivalent.

In the above two definitions of equivalence and in others which have been studied (e.g., in [3]), the definition of interpretation imposes the restriction that the functions should be total. Since, however, under a semiinterpretation (as defined informally above) a program scheme itself is in general a partial function, it appears natural to drop the restriction; we then have, in an imprecise sense which could be made precise, that a program scheme is a functional whose arguments and results alike are partial functions. Another motive for dropping the restriction is that in many standard interpretations the functions are *not* total (e.g., negative radicands, zero divisors, and subroutines that only terminate under certain conditions on their parameters).

The purpose of this paper is to explore several definitions of program scheme equivalence that do not necessarily impose the restriction mentioned above. The work is an extension of [3, 5], uses largely the same notation, and relies on those papers quite strongly, but is self-contained except where it is explicitly stated otherwise.

2. Definitions

We use *location* $L_1, L_2, \ldots$, *function letters* $F_1, F_2, \ldots$, and *predicate* (or *test*) *letters* $T_1, T_2, \ldots$. Each function letter is m-ary for some $m \geqslant 1$, and each test letter is unary.

Remark. We adopt these restrictions to remain consistent with [3]. However, the presence of nonunary test letters makes no difference to any results in this paper. The position with 0-ary functions is rather different. If we allow them, the results of Sections 3–5 are unaffected; one may easily show that having a 0-ary function letter is equivalent to having an extra location letter which is never assigned to. However, Section 6 is no longer valid; in particular Theorem 6.1 fails, as may be shown by using a 0-ary function in place of the extra location L_3 in the proof of Theorem 4.3(i).

Henceforth, in these definitions F, T stand for any function, test letters, and we assume that F has the right number of arguments.

An *assignment* has the form $L_i := F(L_{j_1}, L_{j_2}, ..., L_{j_m})$.

A *test* has the form $T(L_i)$.

A *program scheme* is a finite directed graph, each node of which is labelled by some test and has two successor nodes, or by some assignment and has one successor node, or by STOP and has no successor nodes. One node is designated as the *entry* by an arrow. In the case of a test node, the outgoing arcs are labelled 0, 1. Examples appear in Fig. 3.

We define $\mathscr{S}$ to be the class of all program schemes, $\mathscr{S}_n$ the class with at most n location letters.

A *partial interpretation* I consists of a universe D of individuals, and an assignment:

(i) to each L_i, an individual $L_{iI} \in D$ (the *initial value* of L_i),

(ii) to each m-ary F, a partial function $F_I : D^m \to D$,

(iii) to each T, a partial function $T_I : D \to \{0, 1\}$.

A *total* interpretation I is one in which the F_I, T_I are total functions.

A total (or partial) *finite* interpretation I is one in which the F_I, T_I are total (or partial) and have finite domain. (Thus, a total finite I has finite universe.)

A total (or partial) *recursive* interpretation I is one in which the F_I, T_I are total (or partial) recursive, and the universe is recursive.

Each L_i is an *expression*. If $E_1, ..., E_m$ are expressions so is $F(E_1, ..., E_m)$. These constitute all the expressions.

For each expression E, we define E_I, *the interpretation of* E *under* I, thus: if E is L_i, then E_I is L_{iI}; if E is $F(E_1, ..., E_m)$ then E_I is $F_I(E_{1I}, ..., E_{mI})$ provided each E_{jI} is defined and F_I is defined for these arguments, otherwise E_I is undefined.

An *initial path* in a program scheme P is a finite or infinite path from the entry.

An initial path of P is the *execution sequence* of P under an interpretation I if under I P follows (in an obvious sense) this path and gets no further. The *steps* of an execution sequence are the occurrences of nodes along it, and under a given I each step has associated with it in an obvious way:

(i) (assume k location letters) a member of D^k being the values of $L_1, ..., L_k$ after the step,

(ii) (unless the step is a STOP) an *evaluation* of an F_I or a T_I for certain arguments.

If under I, P's execution sequence is finite and has a last step which is STOP then P *converges* or *stops* under I and $\mathrm{Val}(P_I)$ is defined to be the k-tuple of values of $L_1, ..., L_k$. If the last step has an evaluation $F_I(x_1, ..., x_m)$ or $T_I(x_1)$, $x_i \in D$, then P *sticks* under I on this evaluation. If P's execution sequence under I is infinite, then P *diverges* under I. In the last two cases, $\mathrm{Val}(P_I)$ is undefined.

The following definitions will not be needed until Section 6:

An interpretation (of any type except total finite) is *free* if the universe D is the set of expressions, and if for each F, $F_I(E_1, \ldots, E_m)$ either is undefined *or* takes the value $F(E_1, \ldots, E_m) \in D$.

If I is any interpretation, we define I^0, *its associated free interpretation*, to be that free interpretation in which (i) for each F, $F_{I^0}(E_1, \ldots, E_m)$ *either* is $F(E_1, \ldots, E_m)$ provided $F_I(E_{1I}, \ldots, E_{mI})$ is defined, *or* undefined otherwise, and (ii) for each T, $T_{I^0}(E_1) = T_I(E_{1I})$ or both are undefined.

3. Equivalence Relations

We consider various meanings of the statement "program schemes P and Q behave the same way under all interpretations in a certain class." The meaning has two degrees of freedom:

(i) what constitutes same behaviour, and

(ii) what class of interpretations is considered.

We abbreviate the quoted statement by $P \equiv^{\alpha\beta} Q$, where

(i) α designates the definition of same behaviour, as follows:

α	Definition of Same Behaviour
Empty	"For each appropriate I *either* $\mathrm{Val}(P_I) = \mathrm{Val}(Q_I)$ *or* both are undefined."
*	"For each appropriate I *either* $\mathrm{Val}(P_I) = \mathrm{Val}(Q_I)$ *or* P and Q both diverge *or* P and Q both stick at *some* evaluation."
**	"For each appropriate I *either* $\mathrm{Val}(P_I) = \mathrm{Val}(Q_I)$ *or* P and Q both diverge *or* P and Q both stick at the *same* evaluation."

(ii) β designates the class of interpretations, as follows:

β	Class of Interpretations
t	Total
p	Partial
tr	Total recursive
pr	Partial recursive
tf	Total finite
pf	Partial finite

Thus, for example, $P \equiv^{*pr} Q$ means that under any interpretation in which the functions and tests are partial recursive, *either* P and Q both halt with the same value *or* they both diverge *or* they both stick (but they may stick on different evaluations).

We now define $\mathscr{E}^{\alpha\beta} = \{\langle P, Q\rangle \in \mathscr{S}^2 \mid P \equiv^{\alpha\beta} Q\}$ and $\bar{\mathscr{E}}^{\alpha\beta} = \mathscr{S}^2 - \mathscr{E}^{\alpha\beta}$, and similarly $\mathscr{E}_n^{\alpha\beta}$, $\bar{\mathscr{E}}_n^{\alpha\beta}$, in terms of $\mathscr{S}_n$.

If we note that $P \equiv^{\alpha\beta} Q$ means the same for all α when β is one of t, tr, tf (since a scheme cannot stick on a total interpretation), we have twelve potentially distinct sets $\mathscr{E}^{\alpha\beta}$, which can easily be seen to be partially ordered by inclusion, as shown in Fig. 1.

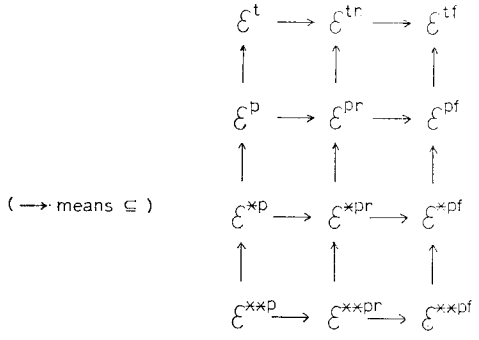

FIG. 1. Inclusions which follow immediately from the definitions of the $\mathscr{E}^{\alpha\beta}$.

The aim of the remainder of this paper is to determine exactly which inclusions between the $\mathscr{E}^{\alpha\beta}$ are proper, and which $\mathscr{E}^{\alpha\beta}$, $\bar{\mathscr{E}}^{\alpha\beta}$ are recursively enumerable. We also answer the same questions for the $\mathscr{E}_n^{\alpha\beta}$ for each n.

4. INCLUSION THEOREMS

In this section we show that of the twelve potentially distinct $\mathscr{E}^{\alpha\beta}$ exactly eight are distinct and are ordered by proper inclusion.

THEOREM 4.1. (i) $\mathscr{E}^{p} \subsetneqq \mathscr{E}^{t}$.
(ii) $\mathscr{E}^{*pf} \subsetneqq \mathscr{E}^{pf}$.
(iii) $\mathscr{E}^{**p} \subsetneqq \mathscr{E}^{*p}$.

Proof. The inclusions have already been shown. To show that they are proper, Fig. 2 gives, for each of the three cases, a pair of program schemes in one class but not in the other. ∎

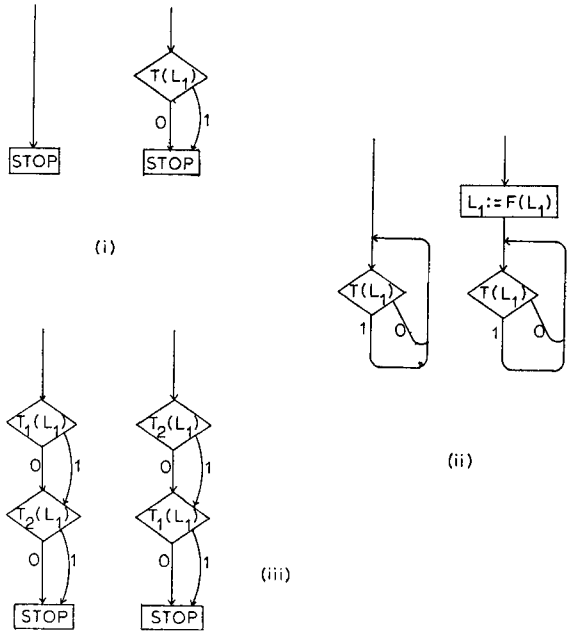

FIG. 2. Examples to show that certain inclusions are proper (Theorem 4.1).

THEOREM 4.2. (i) $\mathscr{E}^{pf} \subseteq \mathscr{E}^{p}$.

(ii) $\mathscr{E}^{**pf} \subseteq \mathscr{E}^{**p}$.

Proof. Consider (ii) first. We prove that $P \not\equiv^{**p} Q \Rightarrow P \not\equiv^{**pf} Q$. Assume $P \not\equiv^{**p} Q$.

Then under some interpretation I, one of the following occurs:

(a) $\mathrm{Val}(P_I)$, $\mathrm{Val}(Q_I)$ are both defined, and unequal.
(b) One of them—say $\mathrm{Val}(P_I)$—is defined, but Q sticks.
(c) Both P and Q stick, and on different evaluations.
(d) One schema—say P—stops, and Q diverges.
(e) One schema—say P—sticks, and Q diverges.

In cases (a), (b), and (c), the behaviour of P and Q is unchanged under the partial finite interpretation I^- formed by restricting the functions and tests of I to those arguments for which they have been successfully evaluated in the (finite) execution sequences of P and Q. Hence in these cases, $P \not\equiv^{**pf} Q$.

In cases (d) and (e), we form the partial finite interpretation I^- by restricting the functions and predicates of I to those arguments for which they have been successfully evaluated in the (finite) execution sequence of P. Then the behaviour of P under I^-

is as under I, while under $I^- Q$ must either diverge or stick. Moreover, if Q sticks, since its execution sequence under I^- must be an initial subsequence of its (divergent) execution sequence under I, it must in case (e) stick at an evaluation different from that at which P sticks under I. Thus, the behaviour of P and Q is different (in the correct sense for $**^{pf}$) under I^-, and hence in cases (d), (e) also $P \not\equiv^{**pf} Q$.

The proof of (i) is similar but only cases (a), (b), and (d) arise. ▮

COROLLARY 4.2.1. *It follows from the results of Section* 3 *that* $\mathscr{E}^{p} = \mathscr{E}^{pr} = \mathscr{E}^{pf}$ *and that* $\mathscr{E}^{**p} = \mathscr{E}^{**pr} = \mathscr{E}^{**pf}$. ▮

Remark. The attempt to prove $\mathscr{E}^{*pf} \subseteq \mathscr{E}^{*p}$ by a similar method breaks down on case (e). The explanation is in the following Theorem.

THEOREM 4.3. (i) $\mathscr{E}^{tr} \subsetneqq \mathscr{E}^{tf}$ *and* $\mathscr{E}^{*pr} \subsetneqq \mathscr{E}^{*pf}$.
(ii) $\mathscr{E}^{t} \subsetneqq \mathscr{E}^{tr}$ *and* $\mathscr{E}^{*p} \subsetneqq \mathscr{E}^{*pr}$.

Proof. (i) We first give a proof from [3] that $\mathscr{E}^{tr} \subsetneqq \mathscr{E}^{tf}$. Consider the scheme Z of Fig. 3. It can easily be seen to diverge on a total I iff $T_I(F_I^n(L_{1I}))$ takes values

$$0, 1, 0, 1, 1, 0, 1, 1, 1, 0,..., \quad \text{for } n = 2, 3, 4,...,$$

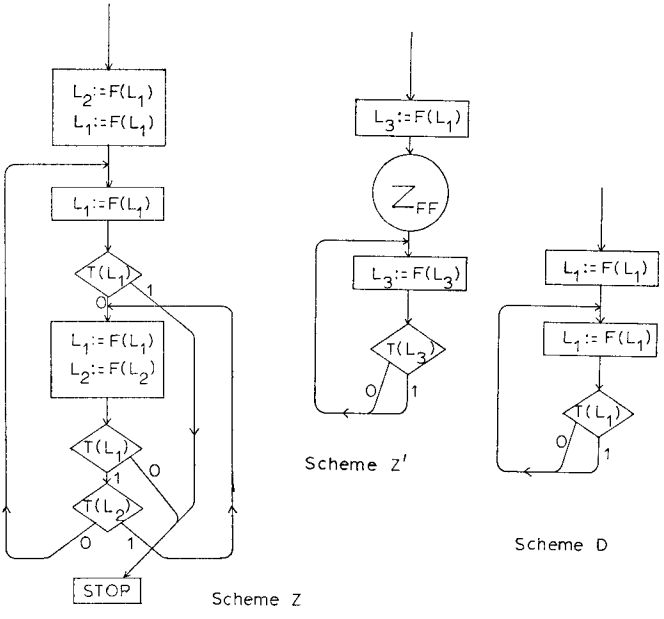

FIG. 3. Examples to show further proper inclusions (Theorem 4.3).

and such an I may be recursive, but cannot be finite, since the sequence is not ultimately periodic. Thus, if P is the scheme consisting of the two assignments $L_1 := F(L_3), L_2 := F(L_3)$ followed by STOP, and Z^+ consists of Z with these two assignments inserted before STOP, we have

$$P \equiv^{tf} Z^+, P \not\equiv^{tr} Z^+$$

and the first part of (i) follows.

Now form Z_{FF} from Z simply by doubling up every assignment instruction (except the initial pair). Then Z_{FF} also converges on all (total) finite interpretations, but diverges on any I for which $T_I(F_I^n(L_{1I}))$ takes the values 0, 1, 0, 1, 1, 0, 1, 1, 1, 0,..., $n = 3, 5, 7,...$(*).

Hence, under any partial finite interpretation the scheme Z' of Fig. 3 *either* (a) diverges in its final loop (*not* in Z_{FF}), *or* (b) sticks.

In case (a), D must also diverge; in case (b), D must also stick (since D evaluates F_I, T_I for all arguments for which Z' does). So $Z' \equiv^{*pf} D$.

But under the partial recursive interpretation for which $T_I(x)$ is defined for values as at (*) above, and undefined for $x = F_I^2(L_{1I})$, Z' diverges and D sticks, so $Z' \not\equiv^{*pr} D$. This proves the second part of (i).

(ii) Paterson [5] exhibits a scheme W, analogous to Z but much more complex, which stops for every recursive interpretation but diverges for some I in which $T_I(F_I^n(L_{1I}))$ takes a nonrecursive sequence $\{\delta_n\}$ of values, $n = 2, 3, 4,...$.

We need not give the details of W here; it is enough to remark that (like Z) it has two locations, one monadic function letter F, one test letter T, that it initializes L_1, L_2 as in Z, and that it evaluates $T_I(F_I^n(L_{1I}))$ for all $n \geqslant 2$ when it diverges. The proof of the first part of (ii) follows by modifying W exactly as Z was modified to prove the first part of (i).

Now we construct W_{FF}, W' analogously to Z_{FF}, Z'; we then have that under any partial recursive interpretation W' *either* (a) diverges in its final loop (*not* in W_{FF}), *or* (b) sticks.

It follows as above that $W' \equiv^{*pr} D$.

However, in the nonpartial recursive interpretation in which $T_I(F_I^{2n-1}(L_{1I})) = \delta_n$ ($n = 2, 3, 4,...$) but $T_I(F_I^2(L_{1I}))$ is undefined, W' diverges and D sticks, so $W' \not\equiv^{*p} D$. This proves the second part of (ii). ∎

The theorems of this section together demonstrate the following ordering of the $\mathscr{E}^{\alpha\beta}$:

COROLLARY 4.3.1.

$$[\mathscr{E}^{**p} = \mathscr{E}^{**pr} = \mathscr{E}^{**pf}] \subsetneqq \mathscr{E}^{*p} \subsetneqq \mathscr{E}^{*pr} \subsetneqq \mathscr{E}^{*pf}$$
$$\subsetneqq [\mathscr{E}^{p} = \mathscr{E}^{pr} = \mathscr{E}^{pf}] \subsetneqq \mathscr{E}^{t} \subsetneqq \mathscr{E}^{tr} \subsetneqq \mathscr{E}^{tf}. \quad ∎$$

5. Unsolvability Results

In this section, we determine which of the $\mathscr{E}^{\alpha\beta}$ and $\bar{\mathscr{E}}^{\alpha\beta}$ are recursively enumerable. We rely strongly on the known results [3, 5] that, of $\mathscr{E}^{t}$, $\mathscr{E}^{tr}$, $\mathscr{E}^{tf}$, and their complements, only $\bar{\mathscr{E}}^{tf}$ is recursively enumerable. In fact, in Theorem 5.2 we rely also on the methods used to obtain these results.

THEOREM 5.1. *$\bar{\mathscr{E}}^{tf}$, $\bar{\mathscr{E}}^{pf}$, $\bar{\mathscr{E}}^{*pf}$, $\bar{\mathscr{E}}^{**pf}$ are recursively enumerable sets.*

Proof. The result for $\bar{\mathscr{E}}^{tf}$ has been mentioned already. Now consider any two schemes P and Q, and an arbitrary partial finite interpretation I. We may suppose that I is given as a (finite) table for each function and predicate, and a set of initial values, one for each register.

Under I, P can only adopt one of a finite number $n_{I,P}$ of states—a state being determined by (a) the value of each register and (b) the node under execution. Thus, P diverges if, and only if, it repeats some state during the first $n_{I,P}+1$ steps of its execution, so the behaviour of P under I (stopping, sticking, or diverging) can be ascertained after a finite amount of computation. (That this amount is bounded by an easily computable bound $n_{I,P}+1$ is unnecessary for our argument.) The same is true of Q.

But we may enumerate all partial finite interpretations $I_1, I_2, \ldots$. Hence, if for example $P \not\equiv^{pf} Q$, we shall certainly find the interpretation under which they behave differently (in the appropriate sense) by executing each of them far enough (in the sense of the previous paragraph) under each I_k for $k = 1, 2, 3, \ldots$. Thus, $\bar{\mathscr{E}}^{pf}$ is recursively enumerable, and similarly for $\bar{\mathscr{E}}^{*pf}$, $\bar{\mathscr{E}}^{**pf}$. ∎

For the next theorem we need another result of Luckham, Park, and Paterson [3, 5]. They construct a class $\mathscr{A} \subseteq \mathscr{S}_2$ of schemata which are not quite standard in that they have two exits, labelled $\underline{a}$, $\underline{r}$ (for accept, reject), and no STOP instruction. They prove that the following sets are not recursively enumerable:

(a) $\{P \in \mathscr{A} \mid P$ diverges under some total interpretation$\}$;

(b) $\{P \in \mathscr{A} \mid P$ diverges under some total recursive interpretation$\}$;

(c) $\{P \in \mathscr{A} \mid P$ never reaches $\underline{a}$ under a total (finite) interpretation$\}$.

The only further information we need about members of $\mathscr{A}$ is that (as Z in Fig. 3) the schemes all initialize L_2 and L_1 to the value $F(L_1)$ and that the value of $F^n(L_1)$ ($n = 2, 3, \ldots$) is tested by T as soon as computed. The construction of $\mathscr{A}$ is also given, in abbreviated form, in Milner [7] in the Appendix.

THEOREM 5.2. (i) *$\bar{\mathscr{E}}^{t}$, $\bar{\mathscr{E}}^{tr}$, $\bar{\mathscr{E}}^{*p}$, $\bar{\mathscr{E}}^{*pr}$ are not recursively enumerable.*

(ii) *For all α, β, $\mathscr{E}^{\alpha\beta}$ is not recursively enumerable.*

Proof. We prove (ii) first. Using result (c), we can see that $P' \equiv^{tf} D$ is not partially decidable, where D, P' are as in Fig. 4 and P' ranges over $\{P' \mid P \in \mathscr{C}\}$.

Now we can show that $P' \equiv^{tf} D \Rightarrow P' \equiv^{**pf} D$ as follows:

Assume $P' \not\equiv^{**pf} D$. But if P' diverges so does D, and if P' sticks D sticks at the same evaluation, since P' tests all values of $F^n(L_1)$, $n \geqslant 2$, as soon as computed. Hence, P' must stop on some partial finite interpretation I, and, therefore, also under some total finite interpretation (by extending all functions arbitarily so that each is defined over the complete finite universe of values computed in P' under I), so it follows that $P' \not\equiv^{tf} D$.

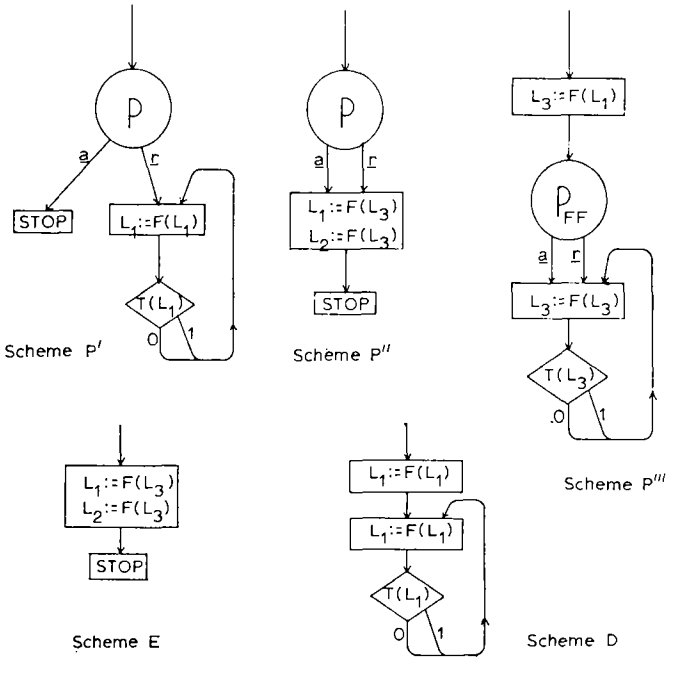

FIG. 4. Examples to show that some $\mathscr{E}^{\alpha\beta}$, $\mathscr{E}^{\alpha\beta}$ are not recursively enumerable (Theorem 5.2)

Now from Corollary 4.3.1 it follows that for any α, β

$$P' \equiv^{**pf} D \Rightarrow P' \equiv^{\alpha\beta} D \Rightarrow P' \equiv^{tf} D,$$

so with the result of the last paragraph we have

$$P' \equiv^{tf} D \Leftrightarrow P' \equiv^{\alpha\beta} D,$$

and so for all α, β, $P' \equiv^{\alpha\beta} D$ is not partially decidable. It follows that $\mathscr{E}^{\alpha\beta}$ is not recursively enumerable.

Now for the Proof of (i). For any $P \in \mathscr{A}$ we have

$$(P \text{ reaches } \underline{a} \text{ or } \underline{r} \text{ under every total } I) \Leftrightarrow P'' \equiv^t E,$$

where P'', E are as in Fig. 4. It follows from result (a) that $P'' \not\equiv^t E$ is not partially decidable, so $\bar{\mathscr{E}}^t$ is not recursively enumerable. The proof for $\bar{\mathscr{E}}^{tr}$ is similar, but is concerned with total *recursive* interpretations and uses result (b).

We now turn to the last two cases of (i).

For any $P \in \mathscr{A}$, we form P_{FF} just as Z_{FF} was formed from Z in Theorem 4.3,viz., by doubling up every assignment instruction except the initializing pair. Then P_{FF} tests the value $F^n(L_1)$ only for $n = 3, 5, 7, \ldots$. We now form P''' as shown in Fig. 4. It may easily be shown that

$$(P \quad \text{reaches } \underline{a} \text{ or } \underline{r} \text{ under every total interpretation}) \Leftrightarrow$$
$$(P_{FF} \text{ reaches } \underline{a} \text{ or } \underline{r} \text{ under every total interpretation}) \Leftrightarrow P''' \equiv^{*p} D.$$

For in particular note that if under some interpretation P''' diverges in P_{FF}, then by making $F_I^2(L_{1I})$ undefined, we can make D stick without altering the behaviour of P'''. It follows from result (a) above that $P''' \not\equiv^{*p} D$ is not partially decidable, so that $\bar{\mathscr{E}}^{*p}$ is not recursively enumerable. The proof for $\bar{\mathscr{E}}^{*pr}$ is similar but is concerned with total (or partial) *recursive* interpretations and uses result (b). ∎

We may summarize the results of this section as the following:

COROLLARY 5.2.1. *Of the eight distinct* $\mathscr{E}^{\alpha\beta}$ (*see Corollary* 4.3.1) *and their complements, exactly four are recursively enumerable, namely,* $\bar{\mathscr{E}}^{tf}$, $\bar{\mathscr{E}}^{pf}$, $\bar{\mathscr{E}}^{*pf}$, $\bar{\mathscr{E}}^{**pf}$.

6. SPECIALIZATION OF RESULTS TO $\mathscr{S}_n$ FOR EACH $n \geqslant 1$

All the results of the last three sections hold when we specialize to $\mathscr{S}_n$ for each $n \geqslant 3$. To go into full details would be tedious, but the reader may verify in particular that (a) when an inclusion was shown to be proper, the counterexample used was in $\mathscr{S}_3$ and (b) when any $\mathscr{E}^{\alpha\beta}$ was shown to be not recursively enumerable, the proof was by exhibiting a nonrecursively enumerable subset of $\mathscr{E}_3^{\alpha\beta}$.

The same is not true for $n = 2$. The main result here is that (Theorem 6.1) we cannot specialize Theorems 4.3 and 5.2(i). We first need some preliminary results and definitions about free interpretations, as defined in Section 2.

If I^0 is the associated free interpretation of I (see the end of Section 2), then we may prove the following, by induction on the length of initial subsequences of execution sequences:

(a) P has the same execution sequence under I^0 as under I.

(b) If at some point in execution of P under I^0 location L_i contains E (an expression) then at the corresponding point in execution of P under I L_i contains E_I.

From these we conclude the following:

FREE INTERPRETATION LEMMA.

$$P \equiv^{\alpha\beta} Q \Leftrightarrow P \equiv^{\alpha\beta^0} Q,$$

when β ranges over $\{t, r, p, pr\}$ and β^0 implies restriction to free β interpretations.

Informally, this may be restated "P, Q behave differently under some β-interpretation iff they behave differently under a *free* β-interpretation."

The following definition supplies something like a finite free interpretation, for use in Theorem 6.1.

DEFINITION. If I is a free interpretation then we define a finite interpretation $I[n]$, *the n-truncation of* I, as follows:

(i) The domain of $I[n]$ is all expressions of depth $\leqslant n$, together with a new individual δ.

(ii) $L_{iI[n]} = L_{iI}$.

(iii) For x_i in the domain of $I[n]$, if $F_I(x_1, ..., x_m)$ is undefined, so is $F_{I[n]}(x_1, ..., x_m)$; otherwise if some x_i has depth n or is δ then $F_{I[n]}(x_1, ..., x_m) = \delta$; otherwise, $F_{I[n]}(x_1, ..., x_m) = F_I(x_1, ..., x_m)$.

(iv) $T_{I[n]}(\delta) = 1$. For $x \neq \delta$, $T_{I[n]}(x) = T_I(x)$ or both are undefined. If $J = I[n]$ for some n, then J is a *truncation* of I. Intuitively an n-truncation of I is like I except that all expressions with depth $>n$ are treated as identical. Clearly if I is total, so is $I[n]$.

THEOREM 6.1.

$$\mathscr{E}_2^{\,t} = \mathscr{E}_2^{tf} \qquad \textit{and} \qquad \mathscr{E}_2^{*p} = \mathscr{E}_2^{*pf}.$$

Proof. It is enough to prove that $P \not\equiv^t Q \Rightarrow P \not\equiv^{tf} Q$ and $P \not\equiv^{*p} Q \Rightarrow P \not\equiv^{*pf} Q$, where P, Q only have two registers L_1, L_2.

Assume first $P \not\equiv^{*p} Q$. Then by the free interpretation Lemma, under some partial free interpretation I^0 one of the following occurs:

(a) P, Q both converge with different results.
(b) One scheme stops, the other sticks.
(c) One scheme—say P—stops, and Q diverges.
(d) One scheme—say P—sticks, and Q diverges.

In cases (a) and (b), the behaviour of P and Q is unchanged under some truncation of I^0 and $P \not\equiv^{*pf} Q$ follows.

In cases (c) and (d) suppose that the maximum depth of evaluation attempted by P

is n. There are two alternatives for Q's execution sequence (denote the depth of expression in L_i after j steps by $\mathrm{dep}(L_i, j)$):

(i) There is some k_0 such that for all $k \geqslant k_0$,

$$\max_{i=1,2}(\mathrm{dep}(L_i, k)) > n.$$

(ii) There is an infinite sequence $k_1, k_2, k_3, \ldots$ such that

$$\mathrm{dep}(L_i, k_j) \leqslant n, \qquad \text{for } i = 1, 2 \qquad \text{and} \qquad j = 1, 2, 3, \ldots$$

In alternative (ii), since the number of expressions with depth $\leqslant n$ is finite and the number of nodes in Q is finite, there must exist $p, m \geqslant 1$ such that the state of Q at step k_p is identical with its state at step k_{p+m}. Thereafter the state sequence is periodic with period m, and no new values (i.e., expressions) are evaluated. Again, therefore, the behaviour of P, Q is unchanged under some truncation of I^0, and $P \not\equiv^{*pf} Q$ follows.

Now assume that alternative (i) holds. There are three subalternatives:

(i)′ For all $k \geqslant k_0$, $\mathrm{dep}(L_1, k) \leqslant n$ and $\mathrm{dep}(L_2, k) > n$.

(i)″ For all $k \geqslant k_0$, $\mathrm{dep}(L_2, k) \leqslant n$ and $\mathrm{dep}(L_1, k) > n$.

(i)‴ For some $k_1 \geqslant k_0$, $\mathrm{dep}(L_i, k_1) > n$, $(i = 1, 2)$.

Consider (i)‴ first. Let $m(>n)$ be the maximum depth of expression computed by Q in the first k_1 steps. We may assume that in I^0 all functions, predicates are defined when arguments have depth $>n$. Then under $I^0[m]$ the behaviour of P is unchanged up to step k_1; and after step k_1, since L_1, L_2 will only contain either δ or expressions of depth $>n$, Q either diverges or stops. Whichever occurs, its behaviour is different from that of P in cases (c) and (d), since in (d) P sticks on an evaluation where depth is $\leqslant n$, and in (c) P's answers have depth $\leqslant n$.

Finally, consider case (i)′. ((i)″ is similar). When $k \geqslant k_0$, since $\mathrm{dep}(L_1, k) < \mathrm{dep}(L_2, k)$, every assignment to L_1 must increase the depth of L_1, so after some step $k_1 \geqslant k_0$ there are no assignments to L_1. Now suppose expressions to unbounded depth are computed in L_2 (otherwise Q computes only finitely many values and we can retain the behaviour of P, Q on some truncation of I^0). Then there is an infinite sequence of steps at each of which Q computes an expression of greater depth than previously, and so for some two such steps k_2, k_3 $(k_1 \leqslant k_2 < k_3)$ Q is at the same node. Suppose the expressions evaluated at these steps are $F_{I^0}(E_1, E_2, \ldots, E_m)$ and $F_{I^0}(E_1', E_2', \ldots, E_m')$, respectively.

Now form I as follows. I is exactly $I^0[\mathrm{dep}(L_2, k_3) - 1]$ except in the value of F_I applied to arguments $E_1', \ldots, E_m'$; in fact, instead of δ we assign it the value of $F_I(E_1, E_2, \ldots, E_m)$, i.e., the expression $F(E_1, E_2, \ldots, E_m)$. Then under I Q will diverge with period $k_3 - k_2$, and P will stop—or stick—as before. Moreover, I is finite.

Therefore we have, for all subalternatives of (i), that $P \not\equiv^{*pf} Q$. Thus, we have shown $P \not\equiv^{*p} Q \Rightarrow P \not\equiv^{*pf} Q$. The proof of $P \not\equiv^{t} Q \Rightarrow P \not\equiv^{tf} Q$ is similar, but slightly simpler since only cases (a) and (c) arise. It also requires the fact that if I^0 is total then so is $I^0[n]$. We omit the details. ▮

COROLLARY 6.1.1.

$$[\mathscr{E}_2^{**p} = \mathscr{E}_2^{**pr} = \mathscr{E}_2^{**pf}] \subsetneq [\mathscr{E}_2^{*p} = \mathscr{E}_2^{*pr} = \mathscr{E}_2^{*pf}]$$
$$\subsetneq [\mathscr{E}_2^{p} = \mathscr{E}_2^{pr} = \mathscr{E}_2^{pf}] \subsetneq [\mathscr{E}_2^{t} = \mathscr{E}_2^{tr} = \mathscr{E}_2^{tf}].$$

Proof. From Corollary 4.3.1, together with Theorem 4.1, noting that the counterexamples in that Theorem are all within $\mathscr{S}_2$. ▮

COROLLARY 6.1.2. *For all* α, β $\bar{\mathscr{E}}_2^{\alpha\beta}$ *is recursively enumerable, but* $\mathscr{E}_2^{\alpha\beta}$ *is not.*

Proof. The first part follows from Theorem 5.1 and Corollary 6.1.1. The second part follows from Theorem 5.2(ii), noting that P', D in that Theorem are in $\mathscr{S}_2$. ▮

Finally, we state the results for $\mathscr{S}_1$.

THEOREM 6.2.

(i) $[\mathscr{E}_1^{**p} = \mathscr{E}_1^{**pr} = \mathscr{E}_1^{**pf}] \subsetneq [\mathscr{E}_1^{*p} = \mathscr{E}_1^{*pr} = \mathscr{E}_1^{*pf}]$
$\subsetneq [\mathscr{E}_1^{p} = \mathscr{E}_1^{pr} = \mathscr{E}_1^{pf}] \subsetneq [\mathscr{E}_1^{t} = \mathscr{E}_1^{tr} = \mathscr{E}_1^{tf}].$

(ii) $\mathscr{E}_1^{\alpha\beta}$ *is recursive for all* α, β.

Proof. (i) The proper inclusions follow from the fact that the counterexamples of Theorem 4.1 are in $\mathscr{S}_1$.

(ii) It is proved in [3] that $\mathscr{E}_1^{t}$ is recursive by translating the equivalence problem for one register program schemes into that for finite automata, which is recursively solvable [6]. This method can be easily adapted in each of the other three cases, and we do not give details. ▮

Remark. All our negative results (i.e., unsolvability, noninclusion) for $\mathscr{S}$, $\mathscr{S}_1$, $\mathscr{S}_2$, have been demonstrated by counterexamples which, except for one, are drawn from the class of schemes with only *one* monadic function letter F and *one* test letter T. The exception is that of Fig. 2(iii): two test letters are used. One may easily construct an example with two registers and only one function, one test letter, and thereby prove Theorem 4.1(iii) under these restrictions; however, it may be shown that this result is *not* true under restriction to one register, one function and one test letter—in fact, the first proper inclusion of Theorem 6.2(i) must be replaced by an equality.

7. Conclusion

Several equivalence relations between program schemata have been studied, in which the interpretations considered may contain partial functions and predicates. These relations turn out to be ordered by inclusion, and while none of them is partially decidable, some of their negations are. One might loosely argue that the partial decidability of $\not\equiv^p$ makes $\equiv^p$ a more natural relation between schemes than $\equiv^t$; however, it is safer simply to say that there are many equivalence relations on program schemata, none of which clearly emerges as the most natural or the most important.

It has also been shown that the class $\mathscr{S}_2$ of program schemes with at most two registers is, perhaps unexpectedly, less structurally rich than the class $\mathscr{S}_3$. This property is rather dependent on the precise definition of a program scheme—we did not, for example, allow 0-ary functions, i.e., constants—but with a different definition one would expect the property to appear in another guise.

Acknowledgment

This paper owes much to the interesting work of Luckham, Park, and Paterson [3] and Paterson [5]. I am also grateful to David Cooper for his encouragement while I carried out this work.

References

1. D. C. Cooper, "Program Scheme Equivalences and Second Order Logic," *in* "Machine Intelligence 4" (B. Meltzer and D. Michie, Eds.), Edinburgh University Press, 1969.
2. D. M. Kaplan, "The Formal Theoretic Analysis of Strong Equivalence for Elemental Programs," Technical Report No. CS101, Computer Science Department, Stanford University, 1966.
3. D. C. Luckham, D. M. R. Park, and M. S. Paterson, "On Formalised Computer Programs," *J. Comp. and Syst. Sci.* **4** (1970).
4. Z. Manna, "Termination of Algorithms," Ph.D. Thesis, Computer Science Department, Carnegie-Mellon University, 1968.
5. M. S. Paterson, "Equivalence Problems in a Model of Computation," Ph. D. Thesis, Cambridge University, 1967.
6. M. O. Rabin and D. Scott, "Finite automata and their decision problems", *IBM J. Res. and Dev.* **3** (1959), 114–125.
7. R. Milner, "Program Schemes and Recursive Function Theory," *in* "Machine Intelligence 5," (D. Michie, Ed.), Edinburgh University Press, 1970.